IN THE CLAIMS

The claims in the case are as follows. No amendments are submitted herein. In the following, it is assumed the amendments submitted in the Appeal Brief of August 14, 2004 have been entered.

1. (previously presented) A method of testing a program having statements, said method comprising the steps of:

a) dividing said program into a plurality of groups such that every statement in the program belongs to at least one of the groups, wherein each of said groups contains a respective sequence of ones of the statements such that all the statements of such a group are executed if at least one statement of said group is executed, and wherein such a group is deemed to be executed if at least one of the statements of the group is executed when the program is executed;

b) determining the ones of the groups that are executed when said program is executed while testing said program;

c) indicating unexecuted ones of the groups based on the ones of the groups that were determined in step b) to have been executed; and

d) enabling a tester to execute said unexecuted groups such that said tester can ensure that all statements in said program are executed at least once.

2. (previously presented) The method of claim 1, further comprising including an extra statement in each of said groups, wherein execution of such an extra statement enables said determining in step b) to identify an executed one of the groups corresponding to said extra statement .

3. (previously presented) The method of claim 2, wherein said extra statements contain respective group identifiers, wherein said determining in step b) further comprises examining such a group identifier to determine a specific one of the groups which has been executed.

4. (original) The method of claim 2, wherein said program is contained in a plurality of programs which in turn are contained in a class of an object oriented environment.

5. (previously presented) The method of claim 4, further comprising the steps of:

grouping a sequence of the groups into a block; and

determining that said block has been executed only if all of the groups of the block are executed.

6. (original) The method of claim 5, wherein said grouping comprises:

determining a language structure present in said plurality of programs;

grouping a subset of groups present in said language structure into a block such that the statements in said language structure are presented as a block to said tester.

7. (original) The method of claim 6, wherein said blocks are defined hierarchically according to the inclusive relationship of language structures in said plurality of programs.

8. (original) The method of claim 7, wherein said language structure comprises one of program delimiters, control structure and loop structure.

9. (previously presented) The method of claim 4, wherein said enabling comprises:

enabling said tester to examine the statements associated with said unexecuted blocks such that said tester can determine arguments which would cause an unexecuted block to be executed;

enabling said tester to enter said determined arguments to cause said unexecuted block to be executed.

10. (previously presented) The method of claim 9, wherein such an argument comprises an instance of another object.

11. (original) The method of claim 10, further comprises:

enabling said tester to instantiate said instance of said another object;

enabling said tester to assign a name to said instance, wherein said tester can enter said name to provide said instance as an argument value.

12. (original) The method of claim 11, further comprising:

receiving a string as an argument; and

determining that said string indicates that said instance is said argument value if said name matches said string.

13. (original) The method of claim 4, further comprising:

enabling said tester to define a macro containing a plurality of program lines;

storing said macro in a database; and

enabling said tester to execute said macro in the middle of testing said plurality of programs.

14. (original) The method of claim 13, wherein said macro is designed to examine the data structures within an instance of an object or to set the values for the variables in the object.

15. (original) The method of claim 4, wherein said dividing, determining, indicating and enabling are performed in a single computer system.

16. (original) The method of claim 4, wherein said object is generated in Java Programming language.

17. (original) The method of claim 4, further comprising:

enabling said tester to load said class;

enabling said tester to instantiate an instance of said class; and

enabling said tester to execute said program on said instance.

18. (previously presented) A computer program product for use with a computer system, said computer program product comprising a computer usable medium having computer readable program code means embodied in said computer usable medium, said computer readable program code means enabling testing of a program having statements, said computer readable program code means comprising:

dividing means for dividing said program into a plurality of groups such that every statement in the program belongs to at least one of the groups, wherein each of said groups contains a respective sequence of ones of the statements such that all the statements of such a group are executed if at least one statement of said group is executed, and wherein such a group is deemed to be executed if at least one of the statements of the group is executed when the program is executed;

determining means for determining the ones of the groups that are executed when said program is executed while testing said program;

indicating means for indicating unexecuted ones of the groups based on said determining; and

enabling means for enabling a tester to execute said unexecuted groups such that said tester can ensure that all statements in said program are executed at least once.

19. (previously presented) The computer program product of claim 18, further comprising including means for including an extra statement in each of said groups, wherein execution of such an extra statement enables said determining means to identify an executed one of the groups corresponding to said extra statement.

20. (previously presented) The computer program product of claim 19, wherein said extra statements contain respective group identifiers, wherein said determining means examines such a group identifier to determine a specific one of the groups which has been executed.

21. (original) The computer program product of claim 19, wherein said program is contained in a plurality of programs which in turn are contained in a class of an object oriented environment.

22. (previously presented) The computer program product of claim 21, further comprising grouping means for grouping a sequence of the groups into a block, and second determining means for determining that said block has been executed only if all of the groups of the block are executed.

23. (previously presented) The computer program product of claim 22, wherein said grouping means comprises:

third determining means for determining a language structure present in said plurality of programs;

a second grouping means for grouping a subset of groups present in said language structure into a block such that the statements in said language structure are presented as a block to said tester.

24. (original) The computer program product of claim 23, wherein said blocks are defined hierarchically according to the inclusive relationship of language structures in said plurality of programs.

25. (previously presented) The computer program product of claim 21, wherein said enabling means comprises:

second enabling means for enabling said tester to examine the statements associated with said unexecuted blocks such that said tester can determine arguments which would cause an unexecuted block to be executed;

third enabling means for enabling said tester to enter said determined arguments to cause said unexecuted block to be executed.

26. (previously presented) The computer program product of claim 25, wherein such an argument comprises an instance of another object, and the computer program product further comprises:

means for enabling said tester to instantiate said instance of said another object;

means for enabling said tester to assign a name to said instance, wherein said tester can enter said name to provide said instance as an argument value.

27. (original) The computer program product of claim 26, further comprising:

means for receiving a string as an argument; and

means for determining that said string indicates that said instance is said argument if said name matches said string.

28. (previously presented) The computer program product of claim 29, wherein said macro is designed to examine the data structures within an instance of an object or to set the values for the variables in the object.

29. (original) The computer program product of claim 21, further comprising:

second enabling means for enabling said tester to define a macro containing a plurality of program lines;

storing means for storing said macro; and

third enabling means for enabling said tester to execute said macro in the middle of testing said plurality of programs.

30. (original) The computer program product of claim 26, further comprising:

means for enabling said tester to load said class;

means for enabling said tester to instantiate an instance of said class; and

means for enabling said tester to execute said program on said instance.

31. (previously presented) A system enabling a tester to test a program having statements, said computer system comprising:

a random access memory (RAM);

a display unit containing a display screen;

an input interface;

a processor dividing said program into a plurality of groups such that every statement in the program belongs to at least one of the groups, wherein each of said groups contains a respective sequence of ones of the statements such that all the statements of such a group are executed if at least one statement of said group is executed, and wherein such a group is deemed to be executed if at least one of the statements of the group is executed when the program is executed,

said processor executing said program in response to instructions received from said input interface,

said processor determining the ones of the groups that are executed when said program is executed,

said processor causing a display to be generated on said display unit, said display indicating unexecuted ones of the groups based on the ones of the groups that were determined to have been executed; and

said processor enabling said tester to execute said unexecuted groups such that said tester can ensure that all statements in said program are executed at least once.

32. (previously presented) The system of claim 31, wherein said processor includes an extra statement in each of said groups, wherein execution of such an extra statement enables said processor to identify an executed one of the groups corresponding to said extra, said computer system further comprising a secondary storage wherein said processor stores said program including said extra statement on said secondary storage.

33. (previously presented) The system of claim 32, wherein said extra statements contain respective group identifiers, wherein said processor examines such a group identifier to determine a specific one of the groups which has been executed.

34. (original) The system of claim 32, wherein said program is contained in a plurality of programs which in turn are contained in a class of an object oriented environment.

35. (previously presented) The system of claim 34, wherein said processor groups a sequence of the groups into a block, and wherein said display indicates that said block has been executed only if all of the groups of the block are executed.

36. (original) The system of claim 35, wherein said processor groups said sequence of groups according to a language structure present in said plurality of programs.

37. (original) The system of claim 36, wherein said blocks are defined hierarchically according to the inclusive relationship of language structures in said plurality of programs.

38. (previously presented) The system of claim 34, wherein said processor receives instructions from said input interface to display the statements associated with said unexecuted blocks, said processor causing the statements to be displayed on said display unit such that said tester can determine arguments which would cause an unexecuted block to be execute.

39. (previously presented) The system of claim 38, wherein such an argument comprises an instance of another object.

40. (original) The system of claim 39, wherein said processor instantiates said instance of another object in response to receiving an instruction to instantiate said instance of said another object, said processor further associating a name associated with said instance of another object, wherein said name is received from said input interface and said tester can enter said name to provide said instance as an argument value.

41. (original) The system of claim 40, wherein said processor receives a string as an argument and determines that said string indicates that said instance is said argument value if said name matches said string.

42. (original) The system of claim 34, wherein said processor receives a plurality of program lines representing a macro, said processor storing said macro in a database, said processor executing said macro in response to receiving an instruction to execute said macro.

43. (original) The system of claim 42, wherein said macro is designed to examine the data structures within an instance of an object or to set the values for the variables in the object.

44. (original) The system of claim 34, wherein said processor loads said class into said RAM in response to receiving an instruction to load said class, said processor further instantiating

an instance of said class in response to receiving another instruction, said processor executing said program on said instance in response to receiving one more instruction.

45. (original) The system of claim 31, wherein said input interface is connected to at least one of a mouse and a key-board.